Lower Bounds for Alternating Online Space Complexity

Nathanaël Fijalkow

Lower Bounds for Alternating Online Space Complexity

Nathanaël Fijalkow

University of Oxford, United Kingdom

Abstract. The notion of online space complexity, introduced by Karp in 1967, quantifies the amount of space required to solve a given problem using an online algorithm, represented by a Turing machine which scans the input exactly once from left to right. In this paper, we study alternating algorithms as introduced by Chandra, Kozen and Stockmeyer in 1976.
We devise a lower bound technique relying on boundedly generated lattices of languages, and give two applications of this technique. The first is a hierarchy theorem for languages of polynomial alternating online space complexity, and the second is a linear lower bound on the alternating online space complexity of the prime numbers written in binary. This second result strengthens a result of Hartmanis and Shank from 1968, which implies an exponentially worse lower bound for the same model.

Keywords: Online Space Complexity, Lower Bounds, Alternating Machines, Hierarchy Theorem, Prime Numbers

1 Online Space Complexity

An *online algorithm* has a restricted access to its input: it scans it exactly once from left to right. The notion of *online computing* has been identified as a fundamental research question in the 80s, and has since then blossomed into several directions with various approaches.

We follow here the approach of Karp [Kar67], who represents online algorithms by online Turing machines, *i.e.* Turing machines whose input tape always moves to the right. We are concerned with *complexity* questions, and in particular about the use of *space* for online algorithms, harkening back to a series of works initiated by Hartmanis and Shank. An impressive result in this line of work is a tight bound on the complexity of checking the primality of a number written in binary for *deterministic* online algorithms [HS69].

We extend this study by considering *alternating* machines as introduced by Chandra, Kozen and Stockmeyer [CS76,Koz76,CKS81]. The notion of alternating machines generalises non-deterministic models where along the computation, the machine makes guesses about the input, and the computation is

accepted if *there exists* a sequence of correct guesses. In other words, these guesses are *disjunctive choices*; the alternating model restores the symmetry by introducing disjunctive and conjunctive choices, resolved by two competing agents, a Prover and a Verifier. Intuitively, the Prover argues that the input is valid and the Verifier challenges this claim.

Our motivations for studying *alternating online machines* is that they form an *expressive* and *robustly implementable* class of online algorithms.

The expressivity is witnessed by the seminal results of Chandra, Kozen and Stockmeyer [CS76,Koz76,CKS81], stating that alternating machines are exponentially more expressive than deterministic ones, which materialises by complexity classes equalities such as APTIME $=$ PSPACE and APSPACE $=$ EXPTIME.

The boolean structure of alternating machines makes them amenable to efficient implementations, taking advantage of parallel and distributed architectures. Such optimisations can be game changers for online algorithms, typically run over very large inputs.

Contributions and organisation of the paper. In this paper, we devise a generic lower bound technique for the alternating online space complexity, based on boundedly generated lattices of languages.

We give the basic definitions, discuss related works and models and show some examples in the remainder of this section. We describe our lower bound technique in Section 2, and give two applications:

- *Hierarchy theorem*: in Section 3, we show a hierarchy theorem: for each natural number ℓ greater than or equal to 2, there exists a language having alternating online space complexity $n^{\ell+1}$ but not n^{ℓ}.
- *Prime numbers*: in Section 4, we look at the language of prime numbers written in binary. The works of Hartmanis and Shank culminated in showing that it does not have subexponential *deterministic* online space complexity [HS69]. We consider the stronger model of *alternating* online algorithms, and first observe that Hartmanis and Shank's techniques imply a *logarithmic* lower bound on the *alternating* online space complexity. Our contribution is to strengthen this result by showing a *linear* lower bound, which is thus an exponential improvement.

1.1 Definitions

We fix an *input alphabet* A, which is a finite set of letters. A *word* is a finite sequence of letters, often denoted $w = w(0)w(1)\cdots w(n-1)$, where $w(i)$'s

are letters from the alphabet A, *i.e.* $w(i) \in A$. We say that w has length n, and denote it $|w|$. The empty word is denoted ε. We denote A^* the set of all words and $A^{\leq n}$ the set of words of length at most n.

For a set E, we denote $\mathcal{B}^+(E)$ the set of positive boolean formulae over E. For instance, if $E = \{p, q, r\}$, an element of $\mathcal{B}^+(E)$ is $p \wedge (q \vee r)$.

Our aim is to prove lower bounds on the space complexity of online Turing machines, *i.e.* which scan their inputs only once from left to right. Following Karp [Kar67], we do not work directly with Turing machines but with a more general model that we simply call machines. Since we are interested in lower bounds, this makes our results stronger. We define alternating machines following Chandra, Kozen and Stockmeyer [CS76,Koz76,CKS81].

Definition 1 (Alternating Machines). *An alternating machine is given by a (potentially infinite) set Q of states, an initial state $q_0 \in Q$, a transition function $\delta : Q \times A \to \mathcal{B}^+(Q)$ and a set of accepting states $F \subseteq Q$.*

To define the semantics of alternating machines, we use acceptance games. Consider an alternating machine $\mathcal{A}$ and an input word w, we define the acceptance game $\mathcal{G}_{\mathcal{A},w}$ as follows: it has two players, Prover and Verifier. The Prover claims that the input word w should be accepted, and the Verifier challenges this claim.

The game starts from the initial state q_0, and with each letter of w read from left to right, a state is chosen through the interaction of the two players. If in a state q and reading a letter a, the new state is obtained using the boolean formula $\delta(q, a)$; Prover chooses which clause is satisfied in a disjunction, and Verifier does the same for conjunctions. A play is won by Prover if it ends up in an accepting state.

The input word w is accepted by $\mathcal{A}$ if Prover has a winning strategy in the acceptance game $\mathcal{G}_{\mathcal{A},w}$. The language recognised by $\mathcal{A}$ is the set of input words accepted by $\mathcal{A}$.

As special cases, a machine is:

- *non-deterministic* if $\delta(q, a)$ is a pure disjunctive formula,
- *universal* if $\delta(q, a)$ is a pure conjunctive formula,
- *deterministic* if $\delta(q, a)$ is an atomic formula, *i.e.* if $\delta : Q \times A \to Q$.

Definition 2 (Online Space Complexity Classes). *Consider a function $f : \mathbb{N} \to \mathbb{N}$ and a language L.*

L is in $\mathrm{Alt}(f)$ if there exists an alternating machine recognising L and a constant C such that for all n in $\mathbb{N}$, the number of different states reachable by some word of length at most n is at most $C \cdot f(n)$.

Similarly, we define $\mathrm{NonDet}\,(f)$ *for non-deterministic machines and* $\mathrm{Det}\,(f)$ *for deterministic machines.*

We denote the function $f : n \mapsto f(n)$ by $f(n)$, making n the implicit variable, so for instance $\mathrm{Alt}\,(n)$ denotes the languages having linear alternating online space complexity. A machine witnessing that L is in $\mathrm{Alt}\,(n)$ is said to use linearly many states (implicitely: in the length of the input word), and the same goes for the usual functions (quadratic, exponential).

For the sake of succinctness, the acronym OSC will be used in place of online space complexity.

The model of machines we defined is actually much more expressive than classical Turing machines. We sketch the ideas for encoding an alternating Turing machine $\mathcal{M}$ with an alternating machine $\mathcal{A}$. The set of states Q of the machine $\mathcal{A}$ is the set of configurations of $\mathcal{M}$. The transitions of the Turing machine $\mathcal{M}$ can be easily simulated by a transition function $\delta : Q \times A \to \mathcal{B}^+(Q)$.

Observe that in our definition of machines, the transition function can be *any* function $\delta : Q \times A \to \mathcal{B}^+(Q)$, without any computability assumption, which is why machines are stronger than Turing machines. In particular, it can easily be seen that machines can describe undecidable languages.

We highlight one unconventional, yet convenient, aspect of our definitions: we count the number of states, which for Turing machines would amount to count the number of configurations. When measuring space for Turing machines, we usually count how many bits are required to describe all possible configurations. It is well known that these two quantities are exponentially related: k bits allow to describe 2^k states, and n states require $\log(n)$ bits to be described. Hence all our lower bounds can be transferred to the realm of Turing machines, with an exponential blow-up.

1.2 Related Works

The research area concerned with *online computing* introduced different models to represent online algorithms. Unlike an *offline algorithm*, which has access to the whole input, an *online algorithm* is presented with its input in a restricted way: it processes it once from left to right.

Related models. We discuss three frameworks that study different aspects of online computing: first the model of *dynamic algorithms*, which generalises online algorithms, second *streaming algorithms*, which take into account both time and space complexity to construct online algorithms, and third the *competitive*

analysis of online algorithms, which aims at quantifying the influence of restricting to online algorithms over offline algorithms.

In the setting of *dynamic algorithms*, the input can go through a series of changes, and the challenge is to construct a data structure together with algorithms for three tasks: initialising, updating and querying the data structure. Whereas for online algorithms, the changes are only insertions, dynamic algorithms also consider deletions, and sometimes more complicated operations.

The seminal paper of Patnaik and Immerman [PI94] introduced the dynamic complexity class DynFO, which is the set of problems whose solutions can be maintained by first-order formulae. This motivated the line of work called dynamic complexity, which has seen recent impressive progress, see for instance [ZS15].

The field of *streaming algorithms* was initiated by a series of papers; Munro and Paterson [MP80], then Flajolet and Martin [FM85], followed by the foundational paper of Alon, Matias and Szegedy [AMS96]. A streaming algorithm has both a limited available memory, much smaller than the input size, and a limited processing time per letter. The challenge there is to use these constrained resources to compute relevant information about the processed input, such as for instance statistics on frequency distributions.

The field of *competitive analysis of online algorithms*, initiated by Sleator and Tarjan [ST85], and by Karp [Kar92], compares the performances between offline and online algorithms, ignoring complexity issues. In this setting, each solution is assigned a real value, assessing its quality. An offline algorithm, having access to the whole input, can select the best solution. An online algorithm, however, has to make choices ignoring part of the input that is still to be read. The question is then whether there exist online algorithms that can perform nearly as good as offline algorithms, up to a competitive ratio.

Related works. The first result about online space complexity together with its definition is due to Karp [Kar67], it states that non-regular languages have at least linear deterministic OSC.

Hartmanis and Shank considered the language of prime numbers written in binary, and showed in [HS69] that it does not have subexponential deterministic OSC. We pursue this question in this paper by consider the alternating OSC of the prime numbers.

Recently, we investigated the online space complexity of probabilistic automata; we substantiated a claim by Rabin [Rab63], by exhibiting a probabilistic automaton which does not have subexponential deterministic OSC [Fij16].

A definition very similar to online space complexity, called automaticity, has been introduced and studied by Shallit and Breitbart [SB96]. The automaticity of a language L is the function $\mathbb{N} \to \mathbb{N}$ which associates to n the size of the smallest automaton which agrees with L on all words of length at most n. The essential difference is that automaticity is a non-uniform notion, as there is a different automaton for each n, whereas online space complexity is uniform, as it considers one machine. For this reason, the two measures behave completely differently. As an argument, consider a language L, and define its exponential padding: $\mathrm{Pad}(L) = \left\{ u\sharp^{2^{|u|}} \mid u \in L \right\}$. It is easy to see that for every language L, its exponential padding $\mathrm{Pad}(L)$ has linear deterministic automaticity. On the other hand, the online space complexity of L and of $\mathrm{Pad}(L)$ are essentially the same, as they are linearly related.

1.3 Complexity Classes and Examples

Denote Reg the class of regular languages, *i.e.* those recognised by finite automata. Then $\mathrm{Det}\,(1) = \mathrm{NonDet}\,(1) = \mathrm{Alt}\,(1) = \mathrm{Reg}$, *i.e.* a language has constant OSC if, and only if, it is regular. Indeed, a machine which uses a constant number of states is essentially a finite automaton, and deterministic, nondeterministic and alternating finite automata are known to be equivalent.

We remark that $\mathrm{Det}\,(\mathrm{Card}(A)^n)$ is the class of all languages. Indeed, consider a language L, we construct a deterministic machine recognising L using exponentially many states. Its set of states is A^*, the initial state is ε and the transition function is defined by $\delta(w, a) = wa$. The set of accepting states is simply L itself. The number of different states reachable by all words of length at most n is the number of words of length at most n, *i.e.* $\frac{\mathrm{Card}(A)^{n+1}-1}{\mathrm{Card}(A)-1}$.

It follows that the maximal OSC of a language is exponential, and the online space complexity classes are relevant for functions smaller than exponential. In particular, the class of languages of polynomial OSC is central.

We now give three examples.

Denote $\textsc{CountEq}_3 = \{w \in \{a, b, c\}^* \mid |w|_a = |w|_b = |w|_c\}$. The notation $|w|_a$ stands for the number of occurrences of the letter a in w.

We construct a deterministic machine recognising this language using quadratically many states. It has two counters that take integers values, initialised to 0 each, which maintain the value $(|w|_a - |w|_b, |w|_a - |w|_c)$. To this end, the letter a acts as $(+1, +1)$, the letter b as $(-1, 0)$, the letter c as $(0, -1)$.

Formally, the set of states is $\mathbb{Z}^2$ and the initial and only accepting state is $(0,0)$. The transitions are:

$$\begin{aligned}\delta((p,q),a) &= (p+1,q+1)\\ \delta((p,q),b) &= (p-1,q)\\ \delta((p,q),c) &= (p,q-1)\end{aligned}$$

After reading the word w, the machine is in the state $(|w|_a - |w|_b, |w|_a - |w|_c)$. This means for a word of length at most n, there are at most $(2n+1)^2$ different states, implying that CountEq$_3$ is in $\mathrm{Det}\left(n^2\right)$.

Denote NotEq $= \{u\sharp v \mid u,v \in \{0,1\}^*, u \neq v\}$.

We construct a non-deterministic machine recognising this language using linearly many states. Note that there are three ways to have $u \neq v$: either v is longer than u, or v is shorter than u, or there exists a position for which they do not carry the same letter.

We focus on the third possibility for the informal explanation. The machine guesses a position in the first word, stores its position p and its letter a, and checks whether the corresponding position in the second word indeed carries a different letter. To this end, after reading the letter $\sharp$, it decrements the position until reaching 1, and checks whether the letter is indeed different from the letter he carries in his state.

Formally, the set of states is $\mathbb{N} \times (A \cup \{\bot\}) \times \{0,1\} \cup \{\top\}$. The first component carries a position, the second component a letter or $\bot$, meaning yet undeclared, and the third component states whether the separator $\sharp$ has been read (1) or not (0). The initial state is $(0,\bot,0)$. The set of accepting states is $\{\top\} \cup \{(p,\bot,1) \mid p \neq 0\}$. The transitions are:

$$\begin{aligned}
&\delta((p,\bot,0),a) = (p+1,a,0) \vee (p+1,\bot,0)\\
&\delta((p,a,0),b) = (p,a,0)\\
&\delta((p,a,0),\sharp) = (p,a,1)\\
&\delta((p,\bot,0),\sharp) = (p,\bot,1)\\
&\delta((p,a,1),b) = (p-1,a,1) && \text{if } p \neq 0\\
&\delta((1,a,1),b) = \top && \text{if } a \neq b\\
&\delta((p,\bot,1),a) = (p-1,\bot,1)\\
&\delta((0,\bot,1),a) = \top\\
&\delta(\top,a) = \top
\end{aligned}$$

If v is longer than u, the state $\top$ will be reached using the second to last transition. If v is shorter than u, a state $(p,\bot,1)$ will be reached with $p \neq 0$. If u and v have the same length and differ on some positions, the state $\top$ will be reached by guessing one such position.

After reading a word of length at most n, the machine can be in one of $(n+1) \cdot (\mathrm{Card}(A)+1) \cdot 2 + 1$ states, thus NOTEQ is in $\mathrm{NonDet}\,(n)$.

Denote $\textsc{FreeGroup} = \left\{u \in (A \cup A^{-1})^* \mid u \text{ is reducible to } \varepsilon\right\}$. There are two reduction rules: $uaa^{-1}v$ reduces to uv, and $ua^{-1}av$ reduces to uv. A word reduces to another if there is a sequence of reductions that leads from the first to the second word.

We construct an alternating machine recognising this language using quadratically many states. To this end, we rely on the following observation. First, define the polarity of a word as the sum of the polarities of each letter, where a in A has polarity 1, and a in A^{-1} has polarity -1, so for instance abb^{-1} has polarity $1+1-1=1$. The following property holds: u of length $2n$ reduces to 1 if, and only if, there exist n letters in u which are mapped to their inverse, with a word of polarity 0 in between. It is proved by induction on n.

The machine has two kinds of states. The first kind, called "guessing states", counts the number of matching letters that the machine has guessed so far, and the total length of the word. The second kind, called "matching states", carries a letter and polarity since this letter has been read, and checks whether the letter can be mapped to its inverse with a word of polarity 0 in between.

The initial state is the guessing state with value 0. Whenever a guessing state reads a new letter, it guesses whether this letter should be among the matching letters or not. If it decides so, it proceeds with two states: one guessing state, whose value has been incremented, and one matching state. Otherwise, it proceeds with the same guessing state, without changing its value.

Formally, the state of states is $(\mathbb{N} \times \mathbb{N}) \cup (A \times \mathbb{Z}) \cup \{\top\}$. The initial state is $(0,0)$. The set of accepting states is $\{\top\} \cup \{(n,2n) \mid n \in \mathbb{N}\}$. The transitions are:

$$\begin{aligned}
&\delta((p,n),a) = (p,n+1) \\
&\qquad \vee ((p+1,n+1) \wedge (a,0)) \\
&\delta((a,0),b) = \top && \text{if } b = a^{-1} \\
&\delta((a,\ell),b) = (a,\ell+1) && \text{if } b \in A \\
&\delta((a,\ell),b) = (a,\ell-1) && \text{if } b \in A^{-1} \\
&\delta(\top,a) = \top
\end{aligned}$$

After reading a word of length at most n, the machine can be in one of $n^2 + \mathrm{Card}(A) \cdot (2n+1) + 1$ states, thus FREEGROUP is in $\mathrm{Alt}\left(n^2\right)$.

2 A Lower Bound Technique

In this section, we develop a generic lower bound technique for the alternating online space complexity. It is based on the size of generating families for some

lattices of languages; we describe it in Subsection 2.1, and a concrete approach to use it, based on query tables, developed in Subsection 2.2. We apply it on an example in Subsection 2.3.

2.1 Boundedly Generated Lattices of Languages

Let L be a language and u a word. The left quotient of L with respect to u is

$$u^{-1}L = \{v \mid uv \in L\}.$$

If u has length at most n, then we say that $u^{-1}L$ has order n.

A lattice of languages is a set of languages closed under union and intersection. Given a family of languages, the lattice it generates is the smallest lattice containing this family.

Theorem 1. *Let L in $\mathrm{Alt}\,(f)$. There exists a constant C such that for all $n \in \mathbb{N}$, there exists a family of $C \cdot f(n)$ languages whose generated lattice contains all the left quotients of L of order n.*

Proof. Let $\mathcal{A}$ be an alternating machine using recognising L witnessing that L is in $\mathrm{Alt}\,(f)$.

Fix n. Denote Q_n the set of states reachable by some word of length at most n; by assumption $\mathrm{Card}(Q_n)$ is at most $C \cdot f(n)$. For q in Q_n, denote $L(q)$ the language recognised by $\mathcal{A}$ taking q as initial state, and $\mathcal{L}_n$ the family of these languages.

We prove by induction over n that all left quotients of L or order n can be obtained as boolean combinations of languages in $\mathcal{L}_n$.

The case $n = 0$ is clear, as $\varepsilon^{-1}L = L = L(q_0)$.

Consider a word w of length $n+1$, denote $w = ua$. We are interested in $w^{-1}L = a^{-1}(u^{-1}L)$, so let us start by $u^{-1}L$. By induction hypothesis, $u^{-1}L$ can be obtained as a boolean combination of languages in $\mathcal{L}_n$: denote $u^{-1}L = \phi(\mathcal{L}_n)$, meaning that ϕ is a boolean formula whose atoms are languages in $\mathcal{L}_n$.

Now consider $a^{-1}\phi(\mathcal{L}_n)$. Observe that the left quotient operation respects both unions and intersections, *i.e.* $a^{-1}(L_1 \cup L_2) = a^{-1}L_1 \cup a^{-1}L_2$ and $a^{-1}(L_1 \cap L_2) = a^{-1}L_1 \cap a^{-1}L_2$. It follows that $w^{-1}L = a^{-1}(\phi(\mathcal{L}_n)) = \phi(a^{-1}\mathcal{L}_n)$; this notation means that the atoms are languages of the form $a^{-1}M$ for M in $\mathcal{L}_n$, *i.e.* $a^{-1}L(q)$ for q in S_n.

To conclude, remark that $a^{-1}L(q)$ can be obtained as a boolean combination of the languages $L(p)$, where p are the states that appear in $\delta(q, a)$. To be more precise, we introduce the notation $\psi(L(\cdot))$, on an example: if $\psi = p \wedge (r \vee s)$, then $\psi(L(\cdot)) = L(p) \wedge (L(r) \vee L(s))$. With this notation, $a^{-1}L(q) =$

$\delta(a, q)(L(\cdot))$. Thus, for q in Q_n, we have that $a^{-1}L(q)$ can be obtained as a boolean combination of languages in $\mathcal{L}_{n+1}$.

Putting everything together, it implies that $w^{-1}L$ can be obtained as a boolean combination of languages in $\mathcal{L}_{n+1}$, finishing the inductive proof. □

2.2 The Query Table Method

Definition 3 (Query Table). *Consider a family of languages $\mathcal{L}$. Given a word w, its profile with respect to $\mathcal{L}$, or $\mathcal{L}$-profile, is the boolean vector stating whether w belongs to L, for each L in $\mathcal{L}$. The size of the query table of $\mathcal{L}$ is the number of different $\mathcal{L}$-profiles, when considering all words.*

For a language L, its query table of order n is the query table of the left quotients of L of order n.

The name query table comes from the following image: the query table of $\mathcal{L}$ is the infinite table whose columns are indexed by languages in $\mathcal{L}$ and rows by words (so, there are infinitely many rows). The cell corresponding to a word w and a language L in $\mathcal{L}$ is the boolean indicating whether w is in L. Thus the $\mathcal{L}$-profile of w is the row corresponding to w in the query table of $\mathcal{L}$.

Lemma 1. *Consider a lattice of languages $\mathcal{L}$ generated by k languages. The query table of $\mathcal{L}$ has size at most 2^k.*

Indeed, there are at most 2^k different profiles with respect to $\mathcal{L}$.

Theorem 2. *Let L in $\mathrm{Alt}\,(f)$. There exists a constant C such that for all $n \in \mathbb{N}$, the query table of L of order n has size at most $2^{C \cdot f(n)}$.*

We show in the next subsection how to use this theorem to prove lower bounds. The proof relies on the following lemma.

Lemma 2. *Consider two lattices of languages $\mathcal{L}$ and $\mathcal{M}$. If $\mathcal{M} \subseteq \mathcal{L}$, then the size of the query table of $\mathcal{M}$ is smaller than or equal to the size of the query table of $\mathcal{L}$.*

Proof. It suffices to observe that the query table of $\mathcal{M}$ is "included" in the query table of $\mathcal{L}$. More formally, consider in the query table of $\mathcal{L}$ the sub-table which consists of rows corresponding to languages in $\mathcal{M}$: this is the query table of $\mathcal{M}$. This implies the claim. □

We now prove Theorem 2. Thanks to Theorem 1, the family of left quotients of L of order n is contained in a lattice generated by a family of size at most $C \cdot f(n)$. It follows from Lemma 2 that the size of the query table of L of order n is smaller than or equal to the size of the query table of a lattice generated by at most $C \cdot f(n)$ languages, which by Lemma 1 is at most $2^{C \cdot f(n)}$.

2.3 A First Application of the Query Table Method

As a first application of our technique, we exhibit a language which has maximal (*i.e.* exponential) alternating OSC. Surprisingly, this language is simple in the sense that it is context-free and definable in Presburger arithmetic.

We say that L has subexponential alternating OSC if $L \in \mathrm{Alt}\,(f)$ for some f such that $f = o(C^n)$ for all $C > 1$. Thanks to Theorem 2, to prove that L does not have subexponential alternating OSC, it is enough to exhibit a constant $C > 1$ such that for infinitely many n, the query table of the left quotients of L of order n has size at least 2^{C^n}.

Theorem 3. *There exists a language which does not have subexponential alternating OSC, yet is both context-free and definable in Presburger arithmetic.*

Proof. Denote

$$L = \left\{ u\sharp u_1\sharp u_2\sharp \cdots \sharp u_k \;\middle|\; \begin{array}{l} u, u_1, \ldots, u_k \in \{0,1\}^*, \\ \exists j \in \{1, \ldots, k\}, u = \overline{u_j} \end{array} \right\}.$$

The notation $\overline{u}$ stands for the reverse of u: formally, $\overline{u} = u(n-1)\cdots u(0)$.

It is easy to see that L is both context-free and definable in Presburger arithmetic (the use of reversed words in the definition of L is only there to make L context-free).

We show that L does not have subexponential alternating OSC. We prove that for all n, the query table of the left quotients of L of order n has size at least 2^{2^n}. Thanks to Theorem 2, this implies the result.

Fix n. Denote by U the set of all words u in $\{0,1\}^n$, it has cardinal 2^n. Consider any subset S of U, we argue that there exists a word w which satisfies that if u in U, then the following equivalence holds:

$$w \in u^{-1}L \iff u \in S.$$

This shows the existence of 2^{2^n} different profiles with respect to the left quotients of order n, as claimed.

Denote $u_1, \ldots, u_{|S|}$ the words in S. Consider

$$w = \sharp\overline{u_1}\sharp\overline{u_2}\sharp \cdots \sharp\overline{u_{|S|}}.$$

The word w clearly satisfies the claim above. □

3 A Hierarchy Theorem for Languages of Polynomial Alternating Online Space Complexity

Theorem 4. *For each $\ell \geq 2$, there exists a language L_ℓ such that:*

- *L_ℓ is in* $\mathrm{Alt}\left(n^\ell\right)$,
- *L_ℓ is not in* $\mathrm{Alt}\left(n^{\ell-1}\right)$.

Consider the alphabet $\{0,1\} \cup \{\Diamond, \sharp\}$.
Let $\ell \geq 2$. Denote

$$L_\ell = \left\{ \Diamond^p u \sharp u_1 \sharp u_2 \sharp \cdots \sharp u_k \;\middle|\; \begin{array}{c} u, u_1, \ldots, u_k \in \{0,1\}^*, \\ j \leq p^\ell \text{ and } u = u_j \end{array} \right\}.$$

Proof.

- The machine has three consecutive phases:
  1. First, a non-deterministic guessing phase while reading $\Diamond^p$, which passes onto the second phase a number j in $\left\{1, \ldots, p^\ell\right\}$.

     Formally, the set of states for this phase is $\mathbb{N}$, the initial state is 0 and the transitions are:
     $$\begin{aligned} &\delta(0, \Diamond) = 1 \\ &\delta(k^\ell, \Diamond) = \bigvee\nolimits_{j \in \left\{1, \ldots, (k+1)^\ell\right\}} j \\ &\delta(p, \Diamond) = p \end{aligned}$$
  2. Second, a universal phase while reading u. Let $n = |u|$. For each i in $\{1, \ldots, n\}$, the machine launches one copy storing the position i, the letter $u(i)$ and the number j guessed in the first phase.

     Formally, the set of states for this phase is $\mathbb{N} \times (\{0,1\} \cup \{\bot\}) \times \mathbb{N}$. The first component is the length of the word read so far (in this phase), the second component stores the letter stored, where the letter $\bot$ stands for undeclared, and the last component is the number j.
     The initial state is $(0, \bot, j)$. The transitions are:
     $$\begin{aligned} &\delta((q, \bot, j), a) = (q+1, \bot, j) \wedge (q, a, j) \\ &\delta((q, a, j), b) = (q, a, j) \end{aligned}$$
     This requires quadratically many states.
  3. Third, a deterministic phase while reading $\sharp u_1 \sharp u_2 \sharp \cdots \sharp u_k$. It starts from a state of the form (q, a, j). It checks whether $u_j(q) = a$. The localisation of the u_j is achieved by decrementing the number j by one each time a letter $\sharp$ is read. While in the corresponding u_j, the localisation of the position q in u_j as achieved by decrementing one position at a time. This requires quadratically many states.

– We now prove the lower bound.
We prove that for all n, the size of the query table of L_ℓ of order $n + 2^{\frac{n}{\ell}}$ is at least 2^{2^n}. Thanks to Theorem 2, this implies that L_ℓ is not in $\mathrm{Alt}\left(n^{\ell-1}\right)$.
Fix n. Denote by U the set of all words u in $\{0,1\}^n$, it has cardinal 2^n.
Observe that $\Diamond^{2^{\frac{n}{\ell}}} u\sharp u_1\sharp u_2\sharp\cdots\sharp u_{2^n}$ belongs to L_ℓ if, and only if, there exists j in $\{1,\ldots,2^n\}$ such that $u = u_j$.
Consider any subset S of U, we argue that there exists a word w which satisfies that if u in U, then the following equivalence holds:

$$w \in \left(\Diamond^{2^{\frac{n}{\ell}}} u\right)^{-1} L \iff u \in S.$$

This shows the existence of 2^{2^n} different profiles with respect to the left quotients of order $n + 2^{\frac{n}{\ell}}$, as claimed.
Denote $u_1,\ldots,u_{|S|}$ the words in S. Consider

$$w = \sharp\overline{u_1}\sharp\overline{u_2}\sharp\cdots\sharp\overline{u_{|S|}}.$$

The word w clearly satisfies the claim above.

4 The Online Space Complexity of Prime Numbers

In this section, we give a lower bound on the alternating online space complexity of the language of prime numbers written in binary:

$$\textsc{Prime} = \{u \in \{0,1\}^* \mid \mathrm{bin}(u) \text{ is prime}\}.$$

By definition $\mathrm{bin}(w) = \sum_{i\in\{0,\ldots,n-1\}} w(i)2^i$; note that the least significant digit is on the left.

The complexity of this language has long been investigated; many efforts have been put in finding upper and lower bounds. Miller gave in 1976 a first conditional polynomial time algorithm, assuming the generalised Riemann hypothesis [Mil76]. In 2002, Agrawal, Kayal and Saxena obtained the same results, but non conditional, *i.e.* not predicated on unproven number-theoretic conjectures [AKS02].

The first lower bounds were obtained by Hartmanis and Shank in 1968, who proved that checking primality requires at least logarithmic deterministic space [HS68], conditionally to number-theoretic assumptions. It was shown by Hartmanis and Berman in 1976 that if the number is presented in unary, then logarithmic deterministic space is necessary and sufficient [HB76].

The best lower bound we know from circuit complexity is due to Allender, Saks and Shparlinski: they proved unconditionally in 2001 that PRIME is not in $\mathrm{AC}^0[p]$ for any prime p [ASS01].

The results above are incomparable to our setting, as we are here interested in online computation. Hartmanis and Shank proved in 1969 that PRIME does not have subexponential deterministic OSC [HS69]. Their result is unconditional, and makes use of Dirichlet's theorem on arithmetic progressions of prime numbers. We show that a corollary of combining their technique and our lower bound approach is that PRIME does not have sublogarithmic alternating OSC.

Our contribution in this section is to extend this result by showing that PRIME does not have sublinear alternating OSC, which is an exponential improvement.

We say that L has sublogarithmic (respectively sublinear) alternating OSC if it is recognised by an alternating machine using f states, where $f = o(\log(n))$ (respectively $f = o(n)$).

Thanks to Theorem 2, to prove that L does not have sublogarithmic (respectively sublinear) alternating OSC, it is enough to exhibit a constant $C > 0$ such that for infinitely many n, the query table of L of order n has size at least $C \cdot n$ (respectively at least $2^{C \cdot n}$).

Theorem 5. *The set of prime numbers written in binary does not have sublinear alternating online space complexity.*

Our result is also unconditional, but it relies on the following advanced theorem from number theory, which can be derived from the results obtained by Maier and Pomerance [MP90]. Note that their results are more general; we simplified the statement to make it both simple and closer to what we actually use.

Simply put, this result proves that in any (reasonable) arithmetic progression and for any degree of isolation, there exists a prime number in this progression, isolated with respect to all prime numbers.

Theorem 6 ([MP90]). *For every arithmetic progression $a + b\mathbb{N}$ such that a and b are coprime, for every N, there exists a number k such that $p = a + b \cdot k$ is the only prime number in $[p - N, p + N]$.*

We prove Theorem 5.

We show that for all $n > 1$, the query table of PRIME of order n has size at least 2^{n-1}. Thanks to Theorem 2, this implies the result.

Fix $n > 1$. Denote by U the set of all words u of length n starting with a 1, *i.e.* odd numbers. It has cardinal 2^{n-1}. Equivalently, we see U as a set of numbers; it contains all the odd numbers smaller than 2^n.

Hartmanis and Shank showed that for two different words u and v in U, the left quotients $u^{-1}\textsc{Prime}$ and $v^{-1}\textsc{Prime}$ are different. It clearly implies that PRIME does not have subexponential deterministic OSC; this is the result stated in [HS69]. The fact that for two different words u and v in U, the left quotients $u^{-1}\textsc{Prime}$ and $v^{-1}\textsc{Prime}$ are different implies a lower bound of $n-1$ on the size of the query table of PRIME of order n. Thus, together with Theorem 2, this proves that PRIME does not have sublogarithmic alternating OSC.

We now prove that the query table of PRIME of order n has size at least 2^{n-1}. To this end, we argue that for all u in U, there exists a word w which satisfies that for all v in S, we have the equivalence between w is in $v^{-1}\textsc{Prime}$ and $u = v$. This means that we construct 2^{n-1} words each having a different profile, implying the claimed lower bound.

Let u in U; denote $a = \mathrm{bin}(u)$. Consider the arithmetic progression $a+2^n\mathbb{N}$; note that a and 2^n are coprime. Thanks to Theorem 6, for $N = 2^n$, there exists a number k such that $p = a + 2^n \cdot k$ is the only prime number in $[p-N, p+N]$. Denote w a word such that $\mathrm{bin}(w) = k$. We show that for all v in U, we have the equivalence between w is in $v^{-1}\textsc{Prime}$ and $u = v$.

Indeed, $\mathrm{bin}(vw) = \mathrm{bin}(v) + 2^n \cdot \mathrm{bin}(w)$. Observe that

$$|\mathrm{bin}(vw) - \mathrm{bin}(uw)| = |\mathrm{bin}(v) - \mathrm{bin}(u)| < 2^n.$$

Since p is the only prime number in $[p-2^n, p+2^n]$, the equivalence follows. This concludes.

We dwell on the possibility of proving stronger longer bounds for the alternating OSC of PRIME. Theorem 6 fleshes out the *sparsity* of prime numbers: it constructs isolated prime numbers in any arithmetic progression, and allowed us to show that the query table of PRIME contains all profiles with all but one boolean value set to false.

To populate the query table of PRIME further, one needs results witnessing the *density* of prime numbers, *i.e.* to prove the existence of clusters of prime numbers. This is in essence the contents of the Twin Prime Conjecture, or more generally of Dickson's conjecture, which are both long-standing open problems in number theory, suggesting that proving better lower bounds is a very challenging objective.

Conclusion

We developed a generic lower bound technique for alternating online Turing machines, and applied it to two problems. The first result is to show that the polynomial hierarchy of alternating online algorithms is infinite. The second result is to give lower bounds on the alternating online space complexity of the language of prime numbers; we show that it is not sublinear, which is an exponential improvement over the previous result. However, the exact complexity is left open; we conjecture that it is not subexponential, but obtaining this result may require major advances in number theory.

References

[AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of Mathematics*, 2:781–793, 2002.

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC'96*, pages 20–29, 1996.

[ASS01] Eric Allender, Michael E. Saks, and Igor Shparlinski. A lower bound for primality. *Journal of Computer and System Sciences*, 62(2):356–366, 2001.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[CS76] Ashok K. Chandra and Larry J. Stockmeyer. Alternation. In *FOCS'76*, pages 98–108, 1976.

[Fij16] Nathanaël Fijalkow. The online space complexity of probabilistic languages. In *LFCS'2016*, pages 106–116, 2016.

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[HB76] Juris Hartmanis and Leonard Berman. On tape bounds for single letter alphabet language processing. *Theoretical Computer Science*, 3(2):213–224, 1976.

[HS68] Juris Hartmanis and H. Shank. On the recognition of primes by automata. *Journal of the ACM*, 15(3):382–389, 1968.

[HS69] Juris Hartmanis and H. Shank. Two memory bounds for the recognition of primes by automata. *Mathematical Systems Theory*, 3(2), 1969.

[Kar67] Richard M. Karp. Some bounds on the storage requirements of sequential machines and turing machines. *Journal of the ACM*, 14(3), 1967.

[Kar92] Richard M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *IFIP'92*, pages 416–429, 1992.

[Koz76] Dexter Kozen. On parallelism in turing machines. In *FOCS'76*, pages 89–97, 1976.

[Mil76] Gary L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13(3):300–317, 1976.

[MP80] J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[MP90] Helmut Maier and Carl Pomerance. Unusually large gaps between consecutive primes. *Transactions of the American Mathematical Society*, 322(1):201–237, 1990.

[PI94] Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. In *PODS'94*, pages 210–221, 1994.

[Rab63] Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

[SB96] Jeffrey Shallit and Yuri Breitbart. Automaticity I: properties of a measure of descriptional complexity. *Journal of Computer and System Sciences*, 53(1):10–25, 1996.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[ZS15] Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Information and Computation*, 240:108–129, 2015.